Campus de l'Université de Montréal
C.P. 6128, succursale Centre ville
Montréal QC H3C 3J7 Canada

Telephone (514) 343 7575
Telecopieur (514) 343 7121

crt@crt.umontreal.ca
www.crt.umontreal.ca

CRT-2002-38

Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation

BIPA
(*BI*LEVEL *P*ROGRAMMING WITH *A*PPROXIMATION METHODS) SOFTWARE GUIDE AND TEST PROBLEMS

by

Benoît Colson

October 2002

PUBLICATION
CRT-2002-38

BIPA
(*BI*LEVEL *P*ROGRAMMING WITH *A*PPROXIMATION METHODS) SOFTWARE GUIDE AND TEST PROBLEMS*

by

Benoît Colson[1]

* This research was supported in part by the Ministère de la Communauté Française (Belgium), the Natural Sciences and Engineering Research Council of Canada (NSERC), and by the Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR).

[1] Département de Mathématiques, Facultés Universitaires Notre-Dame de la Paix, Rempart de la Vierge, 8, B-5000 Namur, Belgique <bc@math.fundp.ac.be>

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2002

103-06396/

Abstract

This paper describes BIPA, a software for solving nonlinear bilevel programming problems. At each iteration, the underlying algorithm computes a linear-quadratic approximation of the original problem around the current iterate. The whole process is embedded in a trust-region framework. E first describe the algorithm before giving details about the implementation and the resulting software and explain how to use it. Finally, a series of test problems is given as well as a complete example with input and output files.

Keywords : bilevel programming, nonlinear programming, trust-region methods, software, test problems.

Résumé

Ce document a pour but de décrire BIPA, un logiciel permettant de résoudre les problèmes de programmation bi-niveaux non linéaires. Lors de chaque itération, l'algorithme calcule une approximation linéaire-quadratique du problème original, et ce autour du point courant. L'ensemble du processus est intégré dans une méthode de type région de confiance. Nous décrivons d'abord l'algorithme avant de donner les détails de son implémentation, puis de présenter le logiciel qui en résulte ainsi que la manière de l'utiliser. Enfin, une série de problèmes tests est fournie, de même qu'un exemple complet avec les fichiers d'entrée et de sortie.

Mots clés : programmation bi-niveaux, programmation non linéaire, méthodes de région de confiance, logiciel, problèmes tests.

1 Introduction

BIPA (*BI*level *P*rogramming with *A*pproximation methods) is a C ANSI code for solving nonlinear bilevel programming problems of the following type:

$$\begin{aligned} \min_{x_1} \quad & F(x_1, x_2) && \text{(1a)} \\ \text{s.t.} \quad & G(x_1) \leq 0 && \text{(1b)} \\ & \min_{x_2} \quad f(x_1, x_2) && \text{(1c)} \\ & \text{s.t.} \quad g(x_1, x_2) \leq 0, && \text{(1d)} \end{aligned}$$

where $F : \mathbb{R}^{n_1 \times n_2} \to \mathbb{R}$, $G : \mathbb{R}^{n_1} \to \mathbb{R}^{m_1}$, $f : \mathbb{R}^{n_1 \times n_2} \to \mathbb{R}$, and $g : \mathbb{R}^{n_1 \times n_2} \to \mathbb{R}^{m_2}$ are twice continuously differentiable functions. We further require f to be convex in x_2 (for a fixed value of x_1).

BIPA is a *trust-region* type method (see *e.g.* Conn, Gould and Toint [7]), where the subproblem consists in solving a sequence of *mixed-integer programs* (MIPs) and *nonlinear optimization problems*. The latter programs are solved using CPLEX [10] routines and DONLP2 [19, 20, 21] respectively.

The current version of BIPA contains the original code, some examples of input files, the complete and adapted version of DONLP2 and an example of *makefile* for testing and running BIPA on a UNIX platform.

The structure of this paper is as follows. We first describe the algorithm underlying BIPA (see Section 2). Section 3 then provides much details about the software, its input files containing parameters and the problem description, the interaction with CLEX and DONLP2 for solving subproblems and the output file with the results of the optimization process. Section 4 gives a complete description of our current set of test problems, one of them being chosen as an example in Section 5 for a step-by-step description of how to solve it with BIPA.

2 Description of the algorithm

This section describes the different steps of the algorithm. More details as well as computational results and a theoretical analysis may be found in [5, 6]. In the sequel, we use $(\bar{x}_1, \bar{x}_2)$ to denote the current point.

1. **Initialization.** The trust-region parameters are initialized, as well as some variables related to stopping criteria (see below):

   - initial trust-region radius : Δ_0,
   - parameters for checking improvement after computation of a new point: η_1, η_2 (with $0 < \eta_1 \leq \eta_2 < 1$),
   - parameters for updating the trust-region radius: γ_1, γ_2 (with $0 < \gamma_1 < 1 < \gamma_2$),
   - parameter for checking closeness of consecutive iterates: ε,
   - maximum allowed number of consecutive unsuccessful iterations: `mxnuns`,
   - minimum value for trust-region radius: $\Delta_{\min}$,
   - maximum number of allowed iterations: $k_{\max}$.

   We also set the iteration counter k to 0 and suppose we are given a first feasible point $(x_1^{(0)}, x_2^{(0)})$.

2. **Computation of the models.** We build a *linear-quadratic* model of problem (1) around the current iterate $(\bar{x}_1, \bar{x}_2)$. More precisely we compute linear approximations of F, G and g, and a quadratic approximation of the lower-level objective f:

$$F_m(x_1,x_2) = F(\bar{x}_1,\bar{x}_2) + \nabla_{x_1}F(\bar{x}_1,\bar{x}_2)^T(x_1-\bar{x}_1) + \nabla_{x_2}F(\bar{x}_1,\bar{x}_2)^T(x_2-\bar{x}_2), \quad (2)$$

$$G_m(x_1) = G(\bar{x}_1) + J_{x_1}G(\bar{x}_1)(x_1-\bar{x}_1), \quad (3)$$

$$\begin{aligned} f_m(x_1,x_2) = {} & f(\bar{x}_1,\bar{x}_2) + \nabla_{x_1}f(\bar{x}_1,\bar{x}_2)^T(x_1-\bar{x}_1) + \nabla_{x_2}f(\bar{x}_1,\bar{x}_2)^T(x_2-\bar{x}_2) \\ & + \frac{1}{2}(x_1-\bar{x}_1)^T\nabla^2_{x_1x_1}f(\bar{x}_1,\bar{x}_2)(x_1-\bar{x}_1) \\ & + (x_1-\bar{x}_1)^T\nabla^2_{x_1x_2}f(\bar{x}_1,\bar{x}_2)(x_2-\bar{x}_2) \\ & + \frac{1}{2}(x_2-\bar{x}_2)^T\nabla^2_{x_2x_2}f(\bar{x}_1,\bar{x}_2)(x_2-\bar{x}_2), \end{aligned} \quad (4)$$

$$g_m(x_1,x_2) = g(\bar{x}_1,\bar{x}_2) + J_{x_1}g(\bar{x}_1,\bar{x}_2)(x_1-\bar{x}_1) + J_{x_2}g(\bar{x}_1,\bar{x}_2)(x_2-\bar{x}_2). \quad (5)$$

These models yield the following linear-quadratic bilevel problem :

$$\begin{aligned} \min_{x_1} \quad & F_m(x_1,x_2) && \text{(6a)} \\ \text{s.t.} \quad & G_m(x_1) \quad \leq 0 && \text{(6b)} \\ & \min_{x_2} \quad f_m(x_1,x_2) && \text{(6c)} \\ & \text{s.t.} \quad g_m(x_1,x_2) \leq 0. && \text{(6d)} \end{aligned}$$

We then introduce the following notations:

$$c_1 = \nabla_{x_1}F(\bar{x}_1,\bar{x}_2) \in \mathbb{R}^{n_1},\ c_2 = \nabla_{x_2}F(\bar{x}_1,\bar{x}_2) \in \mathbb{R}^{n_2}, \quad (7)$$

$$d_1 = \nabla_{x_1}f(\bar{x}_1,\bar{x}_2) \in \mathbb{R}^{n_1},\ d_2 = \nabla_{x_2}f(\bar{x}_1,\bar{x}_2) \in \mathbb{R}^{n_2}, \quad (8)$$

$$A_1 = J_{x_1}G(\bar{x}_1,\bar{x}_2) \in \mathbb{R}^{m_1\times n_1}, \quad (9)$$

$$H_1 = J_{x_1}g(\bar{x}_1,\bar{x}_2) \in \mathbb{R}^{m_2\times n_1},\ H_2 = J_{x_2}g(\bar{x}_1,\bar{x}_2) \in \mathbb{R}^{m_2\times n_2}, \quad (10)$$

$$Q_{11} = \nabla^2_{x_1x_1}f(\bar{x}_1,\bar{x}_2) \in \mathbb{R}^{n_1\times n_1},\ Q_{12} = \nabla^2_{x_1x_2}f(\bar{x}_1,\bar{x}_2) \in \mathbb{R}^{n_1\times n_2}, \quad (11)$$

$$Q_{21} = \nabla^2_{x_2x_1}f(\bar{x}_1,\bar{x}_2) \in \mathbb{R}^{n_2\times n_1},\ Q_{22} = \nabla^2_{x_2x_2}f(\bar{x}_1,\bar{x}_2) \in \mathbb{R}^{n_2\times n_2}, \quad (12)$$

$$r_1 = d_1 - Q_{21}^T\bar{x}_2 - Q_{11}^T\bar{x}_1 \in \mathbb{R}^{n_1},\ r_2 = d_2 - Q_{12}^T\bar{x}_1 - Q_{22}^T\bar{x}_2 \in \mathbb{R}^{n_2}.$$

This allows to modify problem (6) in the following way:

- we first regroup constant terms of (2), (3), (4), and (5) in $\bar{F}$, $\bar{G}$, $\bar{f}$, and $\bar{g}$ respectively :

$$\bar{F} = F(\bar{x}_1,\bar{x}_2) - c_1^T\bar{x}_1 - c_2^T\bar{x}_2,$$

$$\bar{G} = G(\bar{x}_1) - A_1\bar{x}_1,$$

$$\bar{f} = f(\bar{x}_1,\bar{x}_2) - d_1^T\bar{x}_1 - d_2^T\bar{x}_2 + \frac{1}{2}(\bar{x}_1^T\ \bar{x}_2^T)\begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{pmatrix}\begin{pmatrix} \bar{x}_1 \\ \bar{x}_2 \end{pmatrix},$$

$$\bar{g} = g(\bar{x}_1,\bar{x}_2) - H_1\bar{x}_1 - H_2\bar{x}_2;$$

- we then introduce new upper- and lower-level objective and constraint functions as follows :

$$F_{m'}(x_1, x_2) = c_1^T x_1 + c_2^T x_2,$$
$$G_{m'}(x_1) = A_1 x_1,$$
$$f_{m'}(x_1, x_2) = r_1^T x_1 + r_2^T x_2 + \frac{1}{2}(x_1^T \; x_2^T) \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},$$
$$g_{m'}(x_1, x_2) = H_1 x_1 + H_2 x_2.$$

Problem (6) may then be rewritten in the following way :

$$\begin{aligned} \min_{x_1} \quad & F_{m'}(x_1, x_2) && \text{(13a)} \\ \text{s.t.} \quad & G_{m'}(x_1) \leq -\bar{G} && \text{(13b)} \\ & \min_{x_2} \quad f_{m'}(x_1, x_2) && \text{(13c)} \\ & \text{s.t.} \quad g_{m'}(x_1, x_2) \leq -\bar{g}. && \text{(13d)} \end{aligned}$$

3. **Formulation of the mixed-integer programming (MIP) problem.** Starting from problem (13), we now formulate the equivalent mixed-integer programming (MIP) problem to which we add a trust-region type constraint as follows :

$$\begin{aligned} \min \quad & c_1^T x_1 + c_2^T x_2 && \text{(14a)} \\ \text{s.t.} \quad & \|x_1 - \bar{x}_1\| \leq \Delta_k, && \text{(14b)} \\ & A_1 x_1 \leq -\bar{G}, && \text{(14c)} \\ & H_1 x_1 + H_2 x_2 \leq -\bar{g}, && \text{(14d)} \\ & \lambda \geq 0 \in \mathbb{R}^{m_2}, && \text{(14e)} \\ & \lambda_i \leq M z_i, \; i = 1, \ldots, m_2, && \text{(14f)} \\ & [-\bar{g} - H_1 x_1 - H_2 x_2]_i \leq M(1 - z_i), \; i = 1, \ldots, m_2, && \text{(14g)} \\ & z_i \in \{0, 1\}, \; i = 1, \ldots, m_2, && \text{(14h)} \\ & \tfrac{1}{2}(Q_{12}^T + Q_{21}) x_1 + Q_{22} x_2 + H_2^T \lambda + r_2 = 0 \in \mathbb{R}^{n_2}. && \text{(14i)} \end{aligned}$$

Constraint (14b) is the added trust-region constraint and (14c) is primal feasibility of the upper-level problem. Primal feasibility of the lower-level is ensured with (14d)-(14e) together with the complementarity constraint

$$\lambda_i [-\bar{g} - H_1 x_1 - H_2 x_2]_i = 0, \; i = 1, \ldots, m_2,$$

the latter being replaced by (14f), (14g) and (14h). Finally, (14i) is the dual feasibility of the lower-level problem.

4. **Solving MIP.** The `CPXmipopt` routine from the CPLEX Callable Library [10] is used to solve problem (14), whose solution we denote by (x_1^m, x_2^m).

5. **Solving the lower-level problem.** Replacing the upper-level solution x_1^m in the original lower-level problem (1c)-(1d) leads to the following program :

$$\begin{aligned} \min_{x_2} \quad & f(x_1^m, x_2) \\ \text{s.t.} \quad & g(x_1^m, x_2) \leq 0. \end{aligned} \qquad (15)$$

We may solve this problem and compute the *reaction* $x_2^* = x_2(x_1^m)$ with any code for general nonlinear constrained optimization problems. Actually, we solve it using the C ANSI version of Spellucci's DONLP2 code (see [20] and [21] for the description of the underlying algorithm and [19] for the user's guide).

6. **Updating radius and iterate.** We first compute the ratio of achieved reduction versus predicted reduction

$$\rho_k = \frac{F(\bar{x}_1, \bar{x}_2) - F(x_1^m, x_2^*)}{F_m(\bar{x}_1, \bar{x}_2) - F_m(x_1^m, x_2^m)}.$$

If $\rho_k < \eta_1$, the model is inaccurate so we let $(\bar{x}_1, \bar{x}_2)$ unchanged and set $\Delta := \gamma_1 \Delta$. Otherwise we set $\tilde{x}_1 := \bar{x}_1$ (to store the previous iterate) and $(\bar{x}_1, \bar{x}_2) := (x_1^m, x_2^*)$ since the model allowed a sufficient reduction, while the trust-region radius is set to $\Delta := \gamma_2 \Delta$ if $\rho_k > \eta_2$ (unchanged otherwise).

7. **Stopping criteria.** The optimization process is stopped when $\|\bar{x}_1 - \tilde{x}_1\| < \epsilon$ after a successful iteration, or the actual reduction equals the predicted one and the latter is small, or the past mxnuns iterations are all unsuccessful. The algorithm may also stop if the trust-region radius becomes too small (*i.e.* $\Delta_{k+1} < \Delta_{\min}$) or the number of iterations is too large (*i.e.* $k + 1 > k_{\max}$). Other reasons for stopping may occur, *e.g.* the algorithm may fail to solve a subproblem (the complete list of possible reasons for termination is given in Table 1). If none of these conditions is satisfied, set $k := k + 1$ and go to step 2.

Code	Reason for termination
1	Convergence: no significant progress is made
2	Too many consecutive unsuccessful iterations
3	Trust-region radius is too short
4	Maximum allowed number of iterations is reached
5	CPLEX fails to build problem data arrays
6	CPLEX can not open CPLEX environement
7	CPLEX can not turn on screen indicator
8	CPLEX fails to create MIP
9	CPLEX fails to copy problem data
10	CPLEX fails to copy ctype
11	CPLEX fails to write MIP on disk
12	CPLEX fails to optimize MIP
13	CPLEX: MIP objective value is not available
14	CPLEX fails to get optimal integer x
15	CPLEX fails to get optimal slack values
16	Predicted reduction is too small
17	Actual reduction is equal to predicted reduction and is small
18	Predicted reduction is negative (theoretically impossible)

Table 1: Complete list of stopping criteria for BIPA.

3 Software

We now provide details about the implementation of the algorithm described above. This part may be seen as a *user's guide* since it includes the complete description of all input files required to run BIPA (see Section 3.1). We also explain how the MIP (14) is solved using CPLEX (Section 3.2) and how DONLP2 is used to compute the reaction while solving (15) for a fixed value of x_1 (Section 3.3). Finally, we describe the output file produced by BIPA (Section 3.4).

Throughout this section, the sequence `***` is used to denote any numerical value. Also note that the user is expected to write input files taking all typesetting details (tabs, variable names, ...) into account.

3.1 Input files

3.1.1 Parameters for the algorithm

The file `SETUP.DAT` contains values for algorithmic parameters. Any `SETUP.DAT` file must contain 9 lines where parameters are arranged in the following sequence:

```
DELTA0 10.0
DELMIN 1.0e-6
ETA1 0.01
ETA2 0.90
ITMAX 50
GAMMA1 0.6
GAMMA2 1.4
EPSILON 1.0e-6
BIGM 1.0e+2
MXNUNS 5
```

The values shown hereabove correspond to standard values used for the numerical experiments reported in [5]. The user may therefore let this file unchanged for basic tests unless he/she wants to analyze the effects of some of these parameters.

- `DELTA0` is the initial trust-region radius (denoted by Δ_0 in Step 1 of the algorithm),
- `DELMIN` is the minimum allowed value for the trust-region radius (denoted by $\Delta_{\min}$ in Step 1 of the algorithm),
- `ETA1` and `ETA2` correspond to the parameters η_1 and η_2 respectively,
- `ITMAX` is the maximum number of allowed iterations (denoted by k_{max} in Step 1 of the algorithm),
- `GAMMA1` and `GAMMA2` correspond to the parameters γ_1 and γ_2 for updating the trust-region radius,
- `EPSILON` is the parameter ε for checking closeness of two consecutives iterates,
- `BIGM` is the parameter M appearing in equations (14f) and (14g) above,
- `MXNUNS` is the parameter corresponding to the maximum number of allowed unsuccessful iterations.

`ITMAX` and `MXNUNS` are integer variables; all other parameters are read as `float` and then converted to `double`.

3.1.2 Problem parameters

The file `PROBLEM.DAT` must have the following structure:

```
problem_name
N1 ***
N2 ***
M1 ***
M2 ***
X_11 ***
X_12 ***
...  ...
X_21 ***
X_22 ***
...  ...
```

- The string `problem_name` contains at most 25 characters (including the final `\0`),
- `N1` and `N2` are integers representing the number of upper-level and lower-level variables respectively (see n_1 and n_2 in Section 1),
- `M1` and `M2` are integers representing the number of upper-level and lower-level constraints respectively (see m_1 and m_2 in Section 1),
- `X_11`, `X_12`, ... are initial values for the n_1 upper-level variables (`double`),
- `X_21`, `X_22`, ... are initial values for the n_2 lower-level variables (`double`).

3.1.3 Problem functions

The file `prob_eval.c` contains C code corresponding to functions for computing values (gradients, Jacobians, Hessians, ...) related to the functions F, G, f and g appearing in a problem formulated in the same way as (1). In what follows, the vector `x` has `N1` + `N2` components (corresponding to those of x_1 and x_2) and these are stored in `x[0]`, `x[1]`, ..., `x[N1+N2-1]`. All functions described below take such a vector `x` as input.

1. `double evalFu( double *x )`
   evaluates the upper-level objective function value
   $$F(x_1, x_2);$$
2. `void evalGu( double *Gu, double *x )`
   evaluates the upper-level constraint values
   $$G(x_1, x_2) \in \mathbb{R}^{m1}$$
   and stores the result in `Gu[0]`, ..., `Gu[M1-1]`.
3. `double evalfl( double *x )`
   evaluates the lower-level objective function value
   $$f(x_1, x_2).$$

4. `void evalgl( double *gl, double *x )`
   evaluates the lower-level constraint values

$$g(x_1, x_2) \in \mathbb{R}^{m2}$$

   and stores the result in `gl[0]`, ..., `gl[M2-1]`.

5. `void evalFugrad1( double *Fugrad1, double *x )`
   evaluates the n_1 components of the gradient

$$c_1 = \nabla_{x_1} F(x_1, x_2) \in \mathbb{R}^{n_1}$$

   and stores the result in `Fugrad1[0]`, ..., `Fugrad1[N1-1]`.

6. `void evalFugrad2( double *Fugrad2, double *x )`
   evaluates the n_2 components of the gradient

$$c_2 = \nabla_{x_2} F(x_1, x_2) \in \mathbb{R}^{n_2}$$

   and stores the result in `Fugrad2[0]`, ..., `Fugrad2[N2-1]`.

7. `void evalGujac1( double **Gujac1, double *x )`
   evaluates the $m_1 \times n_1$ components of the Jacobian

$$A_1 = J_{x_1} G(\bar{x}_1, \bar{x}_2) \in \mathbb{R}^{m_1 \times n_1}$$

   and stores the result in

$$\begin{array}{ccc} \texttt{Gujac1[0][0]} & \dots & \texttt{Gujac1[0][N1-1]} \\ \vdots & \ddots & \vdots \\ \texttt{Gujac1[M1-1][0]} & \dots & \texttt{Gujac1[M1-1][N1-1]}. \end{array}$$

8. `void evalGujac2( double **Gujac2, double *x )`
   evaluates the $m_1 \times n_2$ components of the Jacobian

$$A_2 = J_{x_2} G(\bar{x}_1, \bar{x}_2) \in \mathbb{R}^{m_1 \times n_2}$$

   and stores the result in

$$\begin{array}{ccc} \texttt{Gujac2[0][0]} & \dots & \texttt{Gujac2[0][N2-1]} \\ \vdots & \ddots & \vdots \\ \texttt{Gujac2[M1-1][0]} & \dots & \texttt{Gujac2[M1-1][N2-1]}. \end{array}$$

   Note that this function is not necessary in our framework since we do not consider the case of joint upper-level constraints, that is constraints depending on both x_1 and x_2 at the upper-level. However, we implemented the function `evalGujac2` for making BIPA ready for a possible future version considering such constraints. With the current version of the algorithm and software, all components of $J_{x_2} G(\bar{x}_1, \bar{x}_2)$ should be set to 0.

9. `void evalflgrad1( double *flgrad1, double *x )`
   evaluates the n_1 components of the gradient

$$d_1 = \nabla_{x_1} f(x_1, x_2) \in \mathbb{R}^{n_1}$$

   and stores the result in `flgrad1[0]`, ..., `flgrad1[N1-1]`.

10. `void evalflgrad2( double *flgrad2, double *x )`
evaluates the n_2 components of the gradient

$$d_2 = \nabla_{x_2} f(x_1, x_2) \in \mathbb{R}^{n_2}$$

and stores the result in `flgrad2[0], ..., flgrad2[N2-1]`.

11. `void evalgljac1( double **gljac1, double *x )`
evaluates the $m_2 \times n_1$ components of the Jacobian

$$H_1 = J_{x_1} g(\bar{x}_1, \bar{x}_2) \in \mathbb{R}^{m_2 \times n_1}$$

and stores the result in

$$\begin{matrix} \texttt{gljac1[0][0]} & \dots & \texttt{gljac1[0][N1-1]} \\ \vdots & \ddots & \vdots \\ \texttt{gljac1[M2-1][0]} & \dots & \texttt{gljac1[M2-1][N1-1]}. \end{matrix}$$

12. `void evalgljac2( double **gljac2, double *x )`
evaluates the $m_2 \times n_2$ components of the Jacobian

$$H_2 = J_{x_2} g(\bar{x}_1, \bar{x}_2) \in \mathbb{R}^{m_2 \times n_2}$$

and stores the result in

$$\begin{matrix} \texttt{gljac2[0][0]} & \dots & \texttt{gljac2[0][N2-1]} \\ \vdots & \ddots & \vdots \\ \texttt{gljac2[M2-1][0]} & \dots & \texttt{gljac2[M2-1][N2-1]}. \end{matrix}$$

13. `void evalflhess11( double **flhess11, double *x )`
evaluates the $n_1 \times n_1$ components of the Hessian

$$Q_{11} = \nabla^2_{x_1 x_1} f(\bar{x}_1, \bar{x}_2) \in \mathbb{R}^{n_1 \times n_1}$$

and stores the result in

$$\begin{matrix} \texttt{flhess11[0][0]} & \dots & \texttt{flhess11[0][N1-1]} \\ \vdots & \ddots & \vdots \\ \texttt{flhess11N1-1][0]} & \dots & \texttt{flhess11[N1-1][N1-1]}. \end{matrix}$$

14. `void evalflhess12( double **flhess12, double *x )`
evaluates the $n_1 \times n_2$ components of the Hessian

$$Q_{12} = \nabla^2_{x_1 x_2} f(\bar{x}_1, \bar{x}_2) \in \mathbb{R}^{n_1 \times n_2}$$

and stores the result in

$$\begin{matrix} \texttt{flhess12[0][0]} & \dots & \texttt{flhess12[0][N2-1]} \\ \vdots & \ddots & \vdots \\ \texttt{flhess12[N1-1][0]} & \dots & \texttt{flhess12[N1-1][N2-1]}. \end{matrix}$$

15. `void evalflhess21( double **flhess21, double *x )`
evaluates the $n_2 \times n_1$ components of the Hessian

$$Q_{21} = \nabla^2_{x_2 x_1} f(\bar{x}_1, \bar{x}_2) \in \mathbb{R}^{n_2 \times n_1}$$

and stores the result in

$$\begin{array}{ccc} \texttt{flhess21[0][0]} & \dots & \texttt{flhess21[0][N1-1]} \\ \vdots & \ddots & \vdots \\ \texttt{flhess21[N2-1][0]} & \dots & \texttt{flhess21[N2-1][N1-1]}. \end{array}$$

16. `void evalflhess22( double **flhess22, double *x )`
evaluates the $n_2 \times n_2$ components of the Hessian

$$Q_{22} = \nabla^2_{x_2 x_2} f(\bar{x}_1, \bar{x}_2) \in \mathbb{R}^{n_2 \times n_2}$$

and stores the result in

$$\begin{array}{ccc} \texttt{flhess22[0][0]} & \dots & \texttt{flhess22[0][N2-1]} \\ \vdots & \ddots & \vdots \\ \texttt{flhess22[N2-1][0]} & \dots & \texttt{flhess22[N2-1][N2-1]}. \end{array}$$

3.1.4 DONLP2 files

The last input file needed to run BIPA is `donlp2_eval.c`. Recall that DONLP2 is used as a subprogram to compute the reaction once the MIP is solved. Roughly speaking, `donlp2_eval.c` contains the same type of information as the file `prob_eval.c`, except that it is adapted to solve the lower-level problem only, assuming that the upper-level variables are fixed. This file is detailed in Section 3.3 further in this text.

3.2 Solving the MIP with CPLEX

As was said earlier, we use the routine `cpxmipopt` from CPLEX to solve the mixed-integer program (14). The file `bipmip.c` contains the implementation of the function `setproblemdata` whose aim is to use the elements of the linear-quadratic model (6) to build the necessary data before running CPLEX.

Function `setproblemdata` takes variables and arrays c_1, c_2, A_1, H_1, H_2, Q_{21}, Q_{22}, Δ, $\bar{G}$, $\bar{g}$, M and r_2 as inputs and uses them to produce the arrays described in Table 2, where

$$\begin{array}{lcl} \texttt{numcols} & = & n_1 + n_2 + 2m_2, \\ \texttt{numrows} & = & 2n_1 + m_1 + 3m_2 + n_2, \\ \texttt{numnz} & = & n_1(2 + m_1 + 2m_2 + n_2) + n_2(m_1 + 2m_2 + n_2) + m_2(1 + n_2) + 2m_2. \end{array}$$

The number of variables of the MIP corresponds to $\texttt{numcols} = n_1 + n_2 + 2m_2$ since in addition to the components of x_1 and x_2 we must also consider the multipliers λ_i ($i = 1, \dots, m_2$) and the binary variables z_j ($j = 1, \dots, m_2$). The structure of the constraints is reproduced in Table 3.2. Basically, function `setproblemdata` follows this scheme to fill in the arrays `matval`, `sense` and `rhs` (empty cells correspond to zeros). More details about the variables mentioned here may be found in the CPLEX documentation [10]. We also refer the interested reader to the example files available with each distribution of CPLEX, and more precisely to the files `mipex1.c`, `mipex2.c` and `mipex3.c`.

Name	Length	Use
`double *obj`	`numcols`	coefficients of the objective function
`double *rhs`	`numrows`	right-hand sides
`char *sense`	`numrows`	constraint type ($\leq$, $\geq$ or $=$)
`int *matbeg`	`numcols`	index array for constraints
`int *matcnt`	`numcols`	index array for constraints
`int *matind`	`numnz`	index array for constraints
`double *matval`	`numnz`	coefficients of the constraints
`double *lb`	`numcols`	lower bounds on variables
`double *ub`	`numcols`	upper bounds on variables
`char *ctype`	`numcols`	constraint type

Table 2: Output arrays for function `setproblemdata`.

Number of rows	Coefficients x_1	x_2	λ_i	z_i	Sense	Right-hand side
n_1	$\begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix}$				$\leq$	$\Delta - \bar{x}_1$
n_1	$\begin{pmatrix} -1 & & \\ & \ddots & \\ & & -1 \end{pmatrix}$				$\leq$	$\Delta - \bar{x}_1$
m_1	A_1				$\leq$	$-\bar{G}$
m_2	H_1	H_2			$\leq$	$-\bar{g}$
m_2			$\begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix}$	$\begin{pmatrix} -M & & \\ & \ddots & \\ & & -M \end{pmatrix}$	$\leq$	0
m_2	$-H_1$	$-H_2$		$\begin{pmatrix} M & & \\ & \ddots & \\ & & M \end{pmatrix}$	$\leq$	$\bar{g} + M$
n_2	$-Q_{21}$	$-Q_{22}$	$-H_2^T$		$\leq$	$-r_2$

Table 3: Structure of the MIP constraints as built by `bipmip.c`.

3.3 Computing the reaction with DONLP2

As we said in Section 3.1 above, the user must also provide a `donlp2_eval.c` file before running BIPA. Recall that the latter uses DONLP2 for computing a solution $x_2^* = x_2(x_1^m)$ of the lower-level problem (15) once a solution (x_1^m, x_2^m) of the MIP is known. The file `donlp2_eval.c` simply contains functions for evaluating the objective, constraints and derivative information of problem (15).

We use the C ANSI version of DONLP2, which was automatically translated from its original Fortran F77 version. Therefore there remain typical Fortran structures and features which must be taken into account when writing `donlp2_eval.c`, together with the fact that one now focusses on the last n_2 components of the vector (x_1, x_2). These topics are listed hereunder.

- In this section, `x` denotes the variable for which DONLP2 computes an optimal value solution to the lower-level problem; in other words, `x` corresponds to x_2 and has therefore n_2 components; note that these components are stored in `x[1]`, ..., `x[n2]` (instead of `x[0]`, ..., `x[n2-1]`, which would have been a more conventional storage in C);
- `xnew[0]`, ..., `xnew[n1-1]` are external variables containing the values of the (upper-level) solution variables of the MIP (x_{11}^m, ..., $x_{1n_1}^m$) as returned by CPLEX; these values are considered by DONLP2 as constant terms in the lower-level objective and constraints.

Also note that DONLP2 assumes that constraints are given under the form

$$g(x) \geq 0,$$

while BIPA is built for constraints of the type

$$g(x) \leq 0.$$

The first task of `donlp2_eval.c` is to initialize some parameters:

- `n` denotes the dimension of the problem – in our case `n = N2`,
- `nh` denotes the number of equality constraints – in the framework of BIPA, it must be set to 0;
- `ng` denotes the number of inequality constraints – in our case, it must be equal to `M2`;
- the initial solution `x[1]`, ..., `x[n]` should be initialized with the values of x_2^m, that is

```
x[1] = xnew[n1];
x[2] = xnew[n1+1];
...
x[n] = xnew[n1+n2-1];
```

- we also suggest to copy the following other instructions (they set other parameters to default values recommended in *DONLP2 users guide* [19]):

```
del0 = 1.00e-1;
tau0 = 0.5e0;
tau  = .1e0;

analyt = TRUE;
epsdif = 0.e0;
nreset = 4;
silent = FALSE;
```

- a final comment is that DONLP2 treats bound constraints in a special way - in short, an array called `gunit` contains details about all inequality constraints and stores coefficients for bound constraints (see [19], page 4, for more details, or our example in Section 5).

The objective function is returned in a pointer `*fx` and its gradient is computed and stored in `gradf[1], ..., gradf[n2]`. The constraints are evaluated in a pointer `*gxi` and the gradients are stored in `gradgi[1], ..., gradgi[n2]`. The use of the `switch` command allows to choose a particular constraint amongst the m_2 possible ones. These computations are performed in the following four functions:

`void ef(x[],*fx)`	evaluates the objective function
`void egradf(x[], gradf[])`	computes the gradient of the objective
`void eg(i, x[], *gxi)`	evaluates the i-th constraint
`void egradg(i, x[], gradgi[])`	computes the gradient of the i-th constraint

All other functions required for DONLP2 to work are empty:

```
void setup(void)
void solchk(void)
void eh(i, x[], *hxi)
void egradh(i, x[], gradhi[])
```

Again, we refer the reader to [19] and our example in Section 5 for more details.

Before concluding this section, we would like to mention the fact that we had to slightly modify the file `donlp2.c` so as to be able to use DONLP2 as a subprogram within BIPA. To be precise, the following two modifications had to be done in the original file `donlp2.c`:

- on line 51, the statement

  ```
  void donlp2(void) {
  ```

  must be replaced with

  ```
  double *donlp2(void) {
  ```

- on line 180, the statement

  ```
  return;
  ```

  must be replaced with

  ```
  return x;
  ```

The version of DONLP2 that comes with BIPA is modified accordingly.

3.4 Output

While running, BIPA produces standard (screen) output with detailed information about the specifications of the problems (name, values of n_1, n_2, m_1 and m_2, ...), the starting values (and their modification after running DONLP2) and each step of the algorithm. Each time CPLEX or DONLP2 is called, the computed solution is printed. Moreover BIPA prints information related to the evaluation of ρ, the ratio of achieved versus predicted reduction.

BIPA also produces a file called `RESULTS.DAT` whose structure is reproduced hereunder. In addition to the information related to the initialization (as above), each iteration has a 1-line summary displaying the iteration number (k), the values of F and f at the current point, the ratio ρ, the trust-region radius Δ_k, a binary variable m indicating whether the step was successful ($m = 0$) or not ($m = 1$), the CPU time necessary for computing the reaction with DONLP2 and the total elapsed time since the first iteration. The values of x_1, x_2, F and f at the solution complete the file.

```
Problem name   : ...
Dimension      : n1  = ...
                 n2  = ...
Initial point  : x11 = ...
                 x12 = ...
                 ...   ...
                 x21 = ...
                 x22 = ...
                 ...   ...
=> objectives  : F   = ...
                 f   = ...
After donlp2   : x11 = ...
                 x12 = ...
                 ...   ...
                 x21 = ...
                 x22 = ...
                 ...   ...
=> objectives  : F   = ...
                 f   = ...

-------------------------------------------------------
It. F         f         Rho       Delta     m Donlp2 Total
    ...       ...                 ...               ...
  1 ...       ...         ...       ...     .  ...    ...
  2 ...       ...         ...       ...     .  ...    ...

-------------------------------------------------------

Solution
 x11 = ...
 x12 = ...
 ...   ...
 x21 = ...
 x22 = ...
 ...   ...
 F   = ...
 f   = ...

Total CPU time :  ...
```

4 Test problems

Table 4 gives an overview of the test problems we used for evaluating the performance of BIPA (see [5]). Some of them are well-known in the literature and involve linear or quadratic functions, others were created by *e.g.* introducing nonlinearities in the former instances, a third class of problems is based on two typical applications of bilevel programming in transportation modelling and planning, namely the network design problem and the toll-setting problem, while the last problem was originally introduced by Outrata [14] in a paper dedicated to mathematical programs with equilibrium constraints or MPECs.

All these problems are coded and they are part of the current version of BIPA. Their complete formulation is given in the next pages. Note that we included two linear-linear problems and one linear-quadratic problem. The latter are not interesting to consider for an assessment of our algorithm (since we compute a linear-quadratic approximation of the bilevel program) but they were used for validating BIPA during the early phases of its development so we included them in our collection for completeness.

Problem type	Problem name	n_1	n_2	m_1	m_2	Reference
Linear-linear	`ShimIshiBard97`	1	1	1	5	Shimizu *et al* [17]
	`Savard89`	2	3	3	6	Savard [15]
Linear-quadratic	`AiyoShim84P2`	2	2	5	6	Aiyoshi and Shimizu [1]
Quadratic-quadratic	`Bard88Ex1`	1	1	1	4	Bard [2]
	`Bard88Ex2`	4	4	9	12	Bard [2]
	`Bard88Ex3`	2	2	3	4	Bard [2]
	`Dempe92`	1	1	0	1	Dempe [8]
	`De Silva78`	2	2	0	4	DeSilva [18]
	`FalkLiu95`	2	2	0	4	Falk and Liu [9]
	`ShimAiyo81P1`	1	1	3	3	Shimizu and Aiyoshi [16]
Nonlinear	`BIPA1`	1	1	3	3	new problem
	`BIPA2`	1	1	1	4	new problem
	`BIPA3`	1	1	2	2	new problem
	`BIPA4`	1	1	2	2	new problem
	`BIPA5`	1	2	1	6	new problem
Network design	`NDP1`	5	5	5	8	see appendix
	`NDP2`	5	5	5	8	see appendix
Toll-setting	`TOLL1`	3	8	3	13	Labbé *et al* [11]
	`TOLL2`	3	18	3	28	new problem
	`TOLL4`	2	4	0	6	Brotcorne [4]
	`TOLL5`	1	4	0	6	Brotcorne [4]
MPEC	`Outrata94P1`	1	2	2	4	Outrata [14]

Table 4: Test problems: data and references.

For problems `NDP` and `TOLL`, we found interesting to provide so-called *conversion tables*. The latter state the relationship between the variables of these problems and their names in the various frameworks used, *i.e.* the general formulation of the bilevel problems (1), the specific formulation of the `NDP` and `TOLL` problems (see (16) and (17) later), the variables used by BIPA and the variables used by DONLP2. Hopefully this may help the user testing these problems when he/she wants to get a better understanding of the results.

4.1 Linear-linear problems

- `ShimIshiBard97`: example 16.1.1 in Shimizu *et al* [17].

$$
\begin{array}{lll}
\min_{x_1} & F(x_1,x_2) & = x_1 - 4x_2 \\
\text{s.t.} & -x_1 \leq 0, & \\
 & \min_{x_2} & f(x_1,x_2) = x_2 \\
 & \text{s.t.} & -x_1 - x_2 + 3 \leq 0, \\
 & & -2x_1 + x_2 \leq 0, \\
 & & 2x_1 + x_2 - 12 \leq 0, \\
 & & -3x_1 + 2x_2 + 4 \leq 0, \\
 & & -x_2 \leq 0.
\end{array}
$$

Starting point : $(x_1^0, x_2^0) = (3, 6)$.

- `Savard89`: example in Savard [15].

$$
\begin{array}{lll}
\min_{x_1} & F(x_1,x_2) & = -8x_{11} - 4x_{12} + 4x_{21} - 40x_{22} - 4x_{23} \\
\text{s.t.} & x_{11} + 2x_{12} & -x_{23} - 1.3 \leq 0, \\
 & -x_{11} \leq 0, & \\
 & -x_{12} \leq 0, & \\
 & \min_{x_2} & f(x_1,x_2) = 2x_{21} + x_{22} + 2x_{23} \\
 & \text{s.t.} & -x_{21} + x_{22} + x_{23} - 1 \leq 0, \\
 & & 4x_{11} - 2x_{21} + 4x_{22} - x_{23} - 2 \leq 0, \\
 & & 4x_{12} + 4x_{21} - 2x_{22} - x_{23} - 2 \leq 0, \\
 & & -x_{21} \leq 0, \\
 & & -x_{22} \leq 0, \\
 & & -x_{23} \leq 0.
\end{array}
$$

Starting point : $(x_{11}^0, x_{12}^0, x_{21}^0, x_{22}^0) = (0.75, 0.75, 0.0, 0.0, 1.0)$.

4.2 Linear-quadratic problem

- `AiyoShim84P2`: problem 2 in Aiyoshi and Shimizu [1].

$$
\begin{array}{lll}
\min_{x_1} & F(x_1,x_2) = 2x_{11} & +2x_{12} - 3x_{21} - 3x_{22} - 60 \\
\text{s.t.} & x_{11} + x_{12} + x_{21} & -2x_{22} - 40 \leq 0, \\
 & x_{11} - 50 \leq 0, & \\
 & -x_{11} \leq 0, & \\
 & x_{12} - 50 \leq 0, & \\
 & -x_{12} \leq 0, & \\
 & \min_{x_2} f(x_1,x_2) & = (x_{21} - x_{11} + 20)^2 + (x_{22} - x_{12} + 20)^2 \\
 & \text{s.t.} & -x_{11} + 2x_{21} + 10 \leq 0, \\
 & & -x_{12} + 2x_{22} + 10 \leq 0, \\
 & & x_{21} - 20 \leq 0, \\
 & & -x_{21} - 10 \leq 0, \\
 & & x_{22} - 20 \leq 0, \\
 & & -x_{22} - 10 \leq 0.
\end{array}
$$

Starting point : $(x_{11}, x_{12}, x_{21}, x_{22}) = (10.0, 10.0, -10.0, -10.0)$.

4.3 Quadratic-quadratic programs

- `Bard88Ex1`: example 1 in Bard [2].

$$
\begin{array}{lll}
\min_{x_1} & F(x_1,x_2) & = (x_1-5)^2+(2x_2+1)^2 \\
\text{s.t.} & -x_1 \leq 0, & \\
 & \min_{x_2} & f(x_1,x_2) = (x_2-1)^2 - 1.5x_1x_2 \\
 & \text{s.t.} & -3x_1+x_2+3 \leq 0, \\
 & & x_1-0.5x_2-4 \leq 0, \\
 & & x_1+x_2-7 \leq 0, \\
 & & -x_2 \leq 0.
\end{array}
$$

Starting point: $(x_1,x_2) = (4,0)$.

- `Bard88Ex2`: example 2 in Bard [2].
The original formulation of this problem involved two independent lower-level problems. We regrouped those two problems so as to form a unique lower-level problem.

$$
\begin{array}{lll}
\min_{x_1} & F(x_1,x_2) & = -(200-x_{21}-x_{23})(x_{21}+x_{23}) - (160-x_{22}-x_{24})(x_{22}+x_{24}) \\
\text{s.t.} & x_{11}+x_{12} & +x_{13}+x_{14} \leq 40, \\
 & 0 \leq x_{11} \leq & 10, \\
 & 0 \leq x_{12} \leq & 5, \\
 & 0 \leq x_{13} \leq & 15, \\
 & 0 \leq x_{14} \leq & 20, \\
 & \min_{x_2} & f(x_1,x_2) = (x_{21}-4)^2 + (x_{22}-13)^2 + (x_{23}-35)^2 + (x_{24}-2)^2 \\
 & \text{s.t.} & 0.4x_{21}+0.7x_{22}-x_{11} \leq 0, \\
 & & 0.6x_{21}+0.3x_{22}-x_{12} \leq 0, \\
 & & 0.4x_{23}+0.7x_{24}-x_{13} \leq 0, \\
 & & 0.6x_{23}+0.3x_{24}-x_{14} \leq 0, \\
 & & 0 \leq x_{21} \leq 20, \\
 & & 0 \leq x_{22} \leq 20, \\
 & & 0 \leq x_{23} \leq 40, \\
 & & 0 \leq x_{24} \leq 40.
\end{array}
$$

Starting point: $(x_{11},x_{12},x_{13},x_{14},x_{21},x_{22},x_{23},x_{24}) = (5,5,15,15,0,0,0,0)$.

- `Bard88Ex3`: example 3 in Bard [2].

$$
\begin{array}{lll}
\min_{x_1} & F(x_1,x_2) = & -x_{11}^2 - 3x_{12} - 4x_{21} + x_{22}^2 \\
\text{s.t.} & x_{11}^2+2x_{12}-4 & \leq 0, \\
 & -x_{11} \leq 0, & \\
 & -x_{12} \leq 0, & \\
 & \min_{x_2} & f(x_1,x_2) = 2x_{11}^2 + x_{21}^2 - 5x_{22} \\
 & \text{s.t.} & -x_{11}^2+2x_{11}-x_{12}^2+2x_{21}-x_{22}-3 \leq 0, \\
 & & -x_{12}-3x_{21}+4x_{22}+4 \leq 0, \\
 & & -x_{21} \leq 0, \\
 & & -x_{22} \leq 0.
\end{array}
$$

Starting point: $(x_{11},x_{12},x_{21},x_{22}) = (0,2,4,1)$.

- `Dempe92`: a problem from Dempe [8].

$$\begin{array}{lll} \min_{x_1} & F(x_1,x_2) & = (x_1-3.5)^2+(x_2+4)^2 \\ \text{s.t.} & \min_{x_2} & f(x_1,x_2)=(x_2-3)^2 \\ & \text{s.t.} & x_2^2-x_1\le 0. \end{array}$$

Starting point: $(x_1,x_2)=(1,1)$.

- `DeSilva78`: a problem from De Silva's PhD thesis [18].

$$\begin{array}{lll} \min_{x_1} & F(x_1,x_2) & = x_{11}^2-2x_{11}+x_{12}^2-2x_{12}+x_{21}^2+x_{22}^2 \\ \text{s.t.} & \min_{x_2} & f(x_1,x_2)=(x_{21}-x11)^2+(x_{22}-x12)^2 \\ & \text{s.t.} & 0.5\le x_{21}\le 1.5 \\ & & 0.5\le x_{22}\le 1.5 \end{array}$$

No starting point is given. Ours are $(x_{11},x_{12},x_{21},x_{22})=(0,0,1,1)$ and $(1,1,1,1)$.

- `FalkLiu95`: second example of Falk and Liu [9]. Same problem as `DeSilva78`, except the upper-level objective which is given by

$$F(x_1,x_2)=x_{11}^2-3x_{11}+x_{12}^2-3x_{12}+x_{21}^2+x_{22}^2.$$

We use the same starting points as those of `DeSilva78` above.

- `ShimAiyo81P1`: a problem from Shimizu and Aiyoshi [16].

$$\begin{array}{lll} \min_{x_1} & F(x_1,x_2) & = x_1^2+(x_2-10)^2 \\ \text{s.t.} & x_1\le 15, & \\ & -x_1+x_2 & \le 0, \\ & -x_1\le 0, & \\ & \min_{x_2} & f(x_1,x_2)=(x_1+2x_2-30)^2 \\ & \text{s.t.} & x_1+x_2\le 20, \\ & & x_2\le 20, \\ & & -x_2\le 0. \end{array}$$

Starting point: $(x_1,x_2)=(5,5)$.

4.4 More general nonlinear problems

- `BIPA1`: new problem

$$\begin{array}{lll} \min_{x_1} & F(x_1,x_2) & = (10-x_1)^3+(10-x_2)^3 \\ \text{s.t.} & x_1\le 15, & \\ & -x_1+x_2 & \le 0, \\ & -x_1\le 0, & \\ & \min_{x_2} & f(x_1,x_2)=(x_1+2x_2-15)^4 \\ & \text{s.t.} & x_1+x_2\le 20, \\ & & x_2\le 20, \\ & & -x_2\le 0. \end{array}$$

Starting point (high point): $(x_1,x_2)=(10,10)$.

- BIPA2: new problem

$$\begin{array}{lll} \min_{x_1} & F(x_1,x_2) & = (x_1-5)^2+(2x_2+1)^2 \\ \text{s.t.} & -x_1 \le 0, & \\ & \min_{x_2} & f(x_1,x_2) = (x_2-1)^2 - 1.5x_1x_2 + x_1^3 \\ & \text{s.t.} & -3x_1+x_2+3 \le 0, \\ & & x_1 - 0.5x_2 - 4 \le 0, \\ & & x_1+x_2-7 \le 0, \\ & & -x_2 \le 0. \end{array}$$

Starting point (high point): $(x_1, x_2) = (4, 0)$.

- BIPA3: new problem

$$\begin{array}{lll} \min_{x_1} & F(x_1,x_2) & = (x_1-5)^4+(2x_2+1)^4 \\ \text{s.t.} & x_1+x_2-4 & \le 0, \\ & -x_1 & \le 0, \\ & \min_{x_2} & f(x_1,x_2) = e^{-x_1+x_2} + x_1^2 + 2x_1x_2 + x_2^2 + 2x_1 + 6x_2 \\ & \text{s.t.} & -x_1+x_2-2 \le 0, \\ & & -x_2 \le 0. \end{array}$$

Starting point (high point): $(x_1, x_2) = (4, 0)$.

- BIPA4: new problem

$$\begin{array}{lll} \min_{x_1} & F(x_1,x_2) = & = x_1^2 + (x_2-10)^2 \\ \text{s.t.} & x_1+2x_2-6 & \le 0, \\ & -x_1 \le 0, & \\ & \min_{x_2} & f(x_1,x_2) = x_1^3 + 2x_2^3 + x_1 - 2x_2 - x_1^2 \\ & \text{s.t.} & -x_1+2x_2-3 \le 0, \\ & & -x_2 \le 0. \end{array}$$

Starting point (high point): $(x_1, x_2) = (1.5, 2.25)$.

- BIPA5: new problem

$$\begin{array}{lll} \min_{x_1} & F(x_1,x_2) & = (x_{11}-x_{22})^4 + (x_{21}-1)^2 + (x_{21}-x_{22})^2 \\ \text{s.t.} & -x_{11} \le 0 & \\ & \min_{x_2} & f(x_1,x_2) = 2x_{11} + e^{x_{21}} + x_{21}^2 + 4x_{21} + 2x_{22}^2 - 6x_{22} \\ & \text{s.t.} & 6x_{11} + x_{21}^2 + e^{x_{22}} - 15 \le 0, \\ & & 5x_{11} + x_{21}^4 - x_{22} - 25 \le 0, \\ & & -x_{21} \le 0, \\ & & x_{21} - 4 \le 0, \\ & & -x_{22} \le 0, \\ & & x_{22} - 2 \le 0. \end{array}$$

Starting point (high point): $(x_{11}, x_{21}, x_{22}) = (1, 1, 1)$.

4.5 Network design problem

The network design problem (see *e.g.* Marcotte [12] or Section 2.2 in Migdalas [13]) consists in modifying a transportation infrastructure with the objective of maximizing social welfare or minimizing design and other system costs.
In this paper, we restrict our attention to the fixed-demand network design problem and more precisely we consider the situation where the capacity of the existing links is modified. Assuming that the network has only one origin-destination pair, this problem may be formulated as follows:

$$\min_{z,x} \quad \sum_{a\in\mathcal{A}} \; [x_a t_a(x_a, z_a) + g_a(z_a)] \tag{16a}$$

$$\text{s.t.} \quad z_a > -1 \quad \forall a \in \mathcal{A} \tag{16b}$$

$$\min_x \quad \sum_{a\in\mathcal{A}} \int_0^{x_a} t_a(u, z_a)\, du \tag{16c}$$

$$\text{s.t.} \quad x_a \geq 0 \;\; \forall a \in \mathcal{A} \tag{16d}$$

$$\sum_{p\in\mathcal{P}} f_p = d \tag{16e}$$

$$x_a = \sum_{p\in\mathcal{P}} \delta_{a,p} f_p \;\; \forall a \in \mathcal{A} \tag{16f}$$

where we use the notation of Table 5 and

$$\delta_{a,p} = \begin{cases} 1 & \text{if link } a \text{ belongs to route } p, \\ 0 & \text{otherwise.} \end{cases}$$

$\mathcal{A}$	:	set of links;
z_a	:	capacity enhancement of link a ($a \in \mathcal{A}$);
x_a	:	traffic flow on link a ($a \in \mathcal{A}$);
t_a	:	travel time function for link a ($a \in \mathcal{A}$);
g_a	:	capacity function for link a ($a \in \mathcal{A}$);
$\mathcal{P}$	:	set of routes;
f_p	:	traffic flow on route p ($p \in \mathcal{P}$);
d	:	traffic demand from origin to destination.

Table 5: Notation for the network design problem.

We focused our study on the network of Figure 1, which is well-known in the traffic modelling literature and was originally presented by Braess [3]. The total traffic flow demand is assumed to be $d = 6$. We also suppose that the link travel time functions are given as follows:

$$t_1(x_1, z_1) = 50 + \frac{x_1}{1+z_1}, \;\; t_2(x_2, z_2) = 10\frac{x_2}{1+z_2}, \;\; t_3(x_3, z_3) = 10 + \frac{x_3}{1+z_3},$$

$$t_4(x_4, z_4) = 10\frac{x_4}{1+z_4}, \;\; t_5(x_5, z_5) = 50 + \frac{x_5}{1+z_5},$$

while the capacity functions are assumed to take the form

$$g_a(z_a) = l_a z_a \; (a \in \mathcal{A})$$

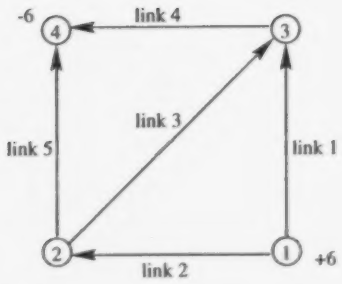

Figure 1: The Braess network considered for problems NDP1 and NDP2.

for fixed parameters l_a ($a \in \mathcal{A}$). There are three possible routes in the network of Figure 1, namely

route 1: link 1 $\rightarrow$ link 4,
route 2: link 2 $\rightarrow$ link 5,
route 3: link 2 $\rightarrow$ link 3 $\rightarrow$ link 4,

and we have the following relationships between route and link flows:

$$\begin{aligned} x_1 &= f_1, \\ x_2 &= f_2 + f_3, \\ x_3 &= f_3, \\ x_4 &= f_1 + f_3, \\ x_5 &= f_2. \end{aligned}$$

This allows to rewrite constraints (16e) and (16f) in terms of the x_i's only:

$$\begin{aligned} x_1 + x_3 + x_5 &= 6, \\ x_2 - x_5 - x_3 &= 0, \\ x_4 - x_1 - x_3 &= 0. \end{aligned}$$

We tested BIPA on the following two problems:

- **Problem** NDP1 ($n_1 = 5, n_2 = 5, m_1 = 5, m_2 = 8$):
  we choose $l_i = 100$ ($i = 0, \ldots, 5$) and the initial values corresponding to the *high point* are

$$z_1 = -0.7, z_2 = 0.0, z_3 = -0.999, z_4 = 0.0, z_5 = -0.7,$$

$$x_1 = 3, x_2 = 3, x_3 = 0, x_4 = 3, x_5 = 3.$$

- **Problem** NDP2: same as NDP1 except that $l_i = 1$ ($i = 0, \ldots, 5$). The *high point* provides the following starting values:

$$z_1 = -0.999, z_2 = 18.0, z_3 = 5.0, z_4 = 18.0, z_5 = -0.999,$$

$$x_1 = 0, x_2 = 6, x_3 = 6, x_4 = 6, x_5 = 0.$$

Table 6 gives the conversion for variables of problems NDP1 and NDP2.

Upper/lower variables in (1)	Names in (16)	BIPA	DONLP2
x_{11}	z_1	`x[0]`	`xnew[0]`
x_{12}	z_2	`x[1]`	`xnew[1]`
x_{13}	z_3	`x[2]`	`xnew[2]`
x_{14}	z_4	`x[3]`	`xnew[3]`
x_{15}	z_5	`x[4]`	`xnew[4]`
x_{21}	x_1	`x[5]`	`x[1]`
x_{22}	x_2	`x[6]`	`x[2]`
x_{23}	x_3	`x[7]`	`x[3]`
x_{24}	x_4	`x[8]`	`x[4]`
x_{25}	x_5	`x[9]`	`x[5]`

Table 6: Conversion table for the variables of problems `NDP1` and `NDP2`.

4.6 Toll-setting problems

Toll-setting problems are another class of models that are frequently formulated as bilevel programs. They represent a situation where an authority or the owners of a highway system are allowed to set tolls on a subset of the links of the network while the network users wish to minimize their travel costs. An optimal toll setting is such that toll levels are not too high – otherwise the users may be detered from using the infrastructure – though still generating profits.
The upper-level variables are the tolls, which we denote by T_a (where a belongs to the subset of tolled links $\mathcal{A}_1 \subset \mathcal{A}$), and the lower-level variables are the link traffic flows, which we still denote by x_a ($a \in \mathcal{A}$) as in the previous example.
The formulation of the toll-setting problem may be written as follows (see problem (TOP) in Labbé, Marcotte and Savard [11]):

$$\max_{T,x} \quad \sum_{a\in\mathcal{A}_1} T_a x_a \tag{17a}$$

$$\text{s.t.} \quad T_a \geq l_a \quad \forall a \in \mathcal{A}_1, \tag{17b}$$

$$\min_x \quad \sum_{a\in\mathcal{A}_1} (c_a + T_a)x_a + \sum_{a\in\mathcal{A}_2} c_a x_a \tag{17c}$$

$$\text{s.t.} \quad \sum_{a\in i^+} x_a^{kl} - \sum_{a\in i^-} x_a^{kl} = 1 \text{ if } i = k, \ -1 \text{ if } i = l, \text{ and 0 otherwise} \tag{17d}$$

$$\forall i \in \mathcal{N}, \forall (k,l) \in \mathcal{O} \times \mathcal{D}$$

$$x_a = \sum_{(k,l)\in\mathcal{O}\times\mathcal{D}} d^{kl} x_a^{kl} \ \forall a \in \mathcal{A} \tag{17e}$$

$$x_a^{kl} \geq 0 \ \forall a \in \mathcal{A}, \ \forall (k,l) \in \mathcal{O} \times \mathcal{D} \tag{17f}$$

where, in addition to the notation introduced in Table 5 above, we use the symbols described in Table 7. The five problem instances we solved with BIPA are built as follows (note that the tolled links are represented with dashed lines on the accompanying figures):

- **Problem** `Toll1` ($n_1 = 3, n_2 = 8, m_1 = 3, m_2 = 13$ – see also Table 8):
  We consider the network and the costs c_a ($a \in \mathcal{A}$) represented on Figure 2. Nodes 1 and 5 constitute its unique origin-destination pair. Other data values include the following items
  - $d^{(1,5)} = 1$;

c_a	:	(fixed) travel cost for link a ($a \in \mathcal{A}$), exclusive of toll;
$\mathcal{A}_2$	:	$\mathcal{A} \backslash \mathcal{A}_1$;
$\mathcal{N}$	:	set of nodes;
i^+	:	set of links exiting from node $i \in \mathcal{N}$;
i^-	:	set of links ending at node $i \in \mathcal{N}$;
$\mathcal{O}$	:	set of origin nodes;
$\mathcal{D}$	:	set of destination nodes;
x_a^{kl}	:	traffic flow from origin k to destination l on link a ($(k,l) \in \mathcal{O} \times \mathcal{D}$, $a \in \mathcal{A}$);
d^{kl}	:	proportion of traffic flow demand between origin k and destination l ($(k,l) \in \mathcal{O} \times \mathcal{D}$);
l_a	:	lower bound on toll for link a ($a \in \mathcal{A}_1$).

Table 7: Additional notation for the toll-setting problem.

- $\mathcal{A}_1 = \{3, 4, 8\}$,
- $\mathcal{A}_2 = \{1, 2, 5, 6, 7\}$,
- upper-level variables: T_3, T_4 and T_8,
- lower-level variables: x_i, $i = 1, \ldots, 8$,
- starting point: $(T_3, T_4, T_8, x_1, \ldots, x_8) = (0, 5, 5, 0, 1, 0, 0, 0, 1, 0, 0)$.

This problem was originally presented in Labbé *et al* [11].

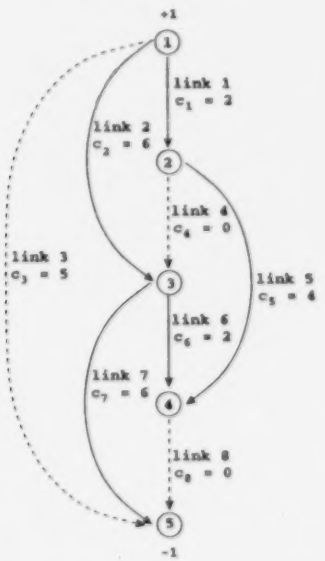

Figure 2: Network for problem `To111`.

Upper/lower variables in (1)	Names in (17)	BIPA	DONLP2
x_{11}	T_3	`x[0]`	`xnew[0]`
x_{12}	T_4	`x[1]`	`xnew[1]`
x_{13}	T_8	`x[2]`	`xnew[2]`
x_{21}	x_1	`x[3]`	`x[1]`
x_{22}	x_2	`x[4]`	`x[2]`
x_{23}	x_3	`x[5]`	`x[3]`
x_{24}	x_4	`x[6]`	`x[4]`
x_{25}	x_5	`x[7]`	`x[5]`
x_{26}	x_6	`x[8]`	`x[6]`
x_{27}	x_7	`x[9]`	`x[7]`
x_{28}	x_8	`x[10]`	`x[8]`

Table 8: Conversion table for the variables of problem `TOLL1`.

- **Problem** `Toll2` ($n_1 = 3, n_2 = 18, m_1 = 3, m_2 = 28$ – see also Table 9):
  The reference network is depicted on Figure 3. There are two origin-destination pairs, namely (O_1, D_1) and (O_2, D_2), with further inputs given as follows:
  - $d^{(O_1,D_1)} = d^{(O_2,D_2)} = 1$;
  - $\mathcal{A}_1 = \{1, 2, 3\}$,
  - $\mathcal{A}_2 = \{4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$,
  - upper-level variables: T_1, T_2 and T_3,
  - lower-level variables: x_i^1, x_i^2 $(i = 1, 2, 3)$ and x_j $(j = 4, \ldots, 15)$, where x_i^k denotes the proportion of flow on link i related to the demand of the k-th origin-destination pair,
  - starting point: $(T_1, T_2, T_3) = (0, 0, 0)$ and $(x_1^1, x_1^2, x_2^1, x_2^2, x_3^1, x_3^2, x_4, \ldots, x_{15}) = (0, 0, 0, 10, 1, 0, 1, 0, 0, 0, 10, 0, 0, 0, 1, 0, 0, 10)$.

- **Problem** `TOLL3`: same as `TOLL2` except that the demand for O-D pair (O_2, D_2) is 10 instead of 1.
  The starting point is $(T_1, T_2, T_3) = (0, 0, 0)$ and $(x_1^1, x_1^2, x_2^1, x_2^2, x_3^1, x_3^2, x_4, \ldots, x_{15}) = (0, 0, 0, 10, 1, 0, 1, 0, 0, 0, 10, 0, 0, 0, 1, 0, 0, 10)$.

- **Problem** `Toll4` ($n_1 = 2, n_2 = 4, m_1 = 0, m_2 = 6$ – see also Table 10):
  This corresponds to an example (Figure 4.3, page 123) in Brotcorne [4], whose network is depicted on Figure 4 and contains two O-D pairs, $(1, 2)$ and $(3, 4)$.
  - $d^{(1,2)} = d^{(5,6)} = 1$;
  - $\mathcal{A}_1 = \{4, 6\}$,
  - $\mathcal{A}_2 = \{1, 2, 3, 5, 7\}$,
  - upper-level variables: T_4 and T_6,
  - lower-level variables: $x_1, x_7, x_2 = x_4^1 = x_3$ and $x_5 = x_4^2 = x_6$,
  - starting point: $(T_4, T_6, x_1, x_2, x_5, x_7) = (0, 0, 0, 0, 0, 0)$.

  Note that this problem does not involve lower-bound constraints of the type (17b), therefore allowing for negative tolls.

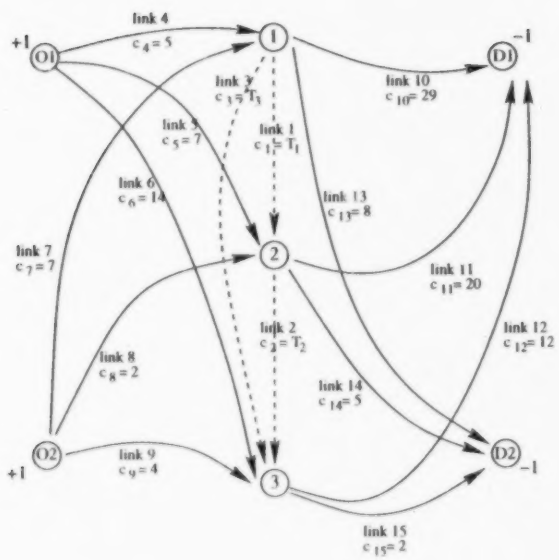

Figure 3: Network for problem `To112`.

Upper/lower variables in (1)	Names in (17)	BIPA	DONLP2
x_{11}	T_1	`x[0]`	`xnew[0]`
x_{12}	T_2	`x[1]`	`xnew[1]`
x_{13}	T_3	`x[2]`	`xnew[2]`
x_{21}	x_1^1	`x[3]`	`x[1]`
x_{22}	x_1^2	`x[4]`	`x[2]`
x_{23}	x_2^1	`x[5]`	`x[3]`
x_{24}	x_2^2	`x[6]`	`x[4]`
x_{25}	x_3^1	`x[7]`	`x[5]`
x_{26}	x_3^2	`x[8]`	`x[6]`
x_{27}	x_4	`x[9]`	`x[7]`
x_{28}	x_5	`x[10]`	`x[8]`
x_{28}	x_6	`x[11]`	`x[9]`
x_{28}	x_7	`x[12]`	`x[10]`
x_{28}	x_8	`x[13]`	`x[11]`
x_{28}	x_9	`x[14]`	`x[12]`
x_{28}	x_{10}	`x[15]`	`x[13]`
x_{28}	x_{11}	`x[16]`	`x[14]`
x_{28}	x_{12}	`x[17]`	`x[15]`
x_{28}	x_{13}	`x[18]`	`x[16]`
x_{28}	x_{14}	`x[19]`	`x[17]`
x_{28}	x_{15}	`x[20]`	`x[18]`

Table 9: Conversion table for the variables of problems `TOLL2` and `TOLL3`.

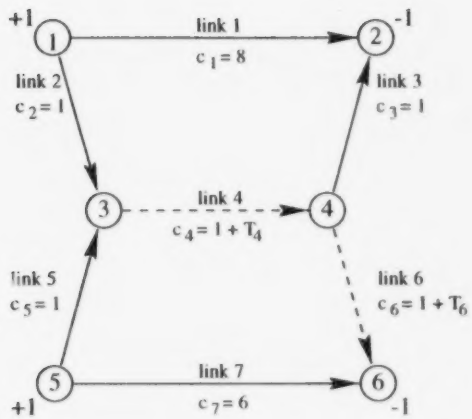

Figure 4: Network for problem `Toll4`.

Upper/lower variables in (1)	Names in (17)	BIPA	DONLP2
x_{11}	T_4	`x[0]`	`xnew[0]`
x_{12}	T_6	`x[1]`	`xnew[1]`
x_{21}	x_1	`x[2]`	`x[1]`
x_{22}	$x_2 = x_4^1 = x_3$	`x[3]`	`x[2]`
x_{23}	$x_5 = x_4^2 = x_6$	`x[4]`	`x[3]`
x_{24}	x_7	`x[5]`	`x[4]`

Table 10: Conversion table for the variables of problem `TOLL4`.

- **Problem** `Toll5` ($n_1 = 1, n_2 = 4, m_1 = 0, m_2 = 6$ – see also Table 11):
  This is another example from Brotcorne [4] (Figure 4.4, page 125), corresponding to the network of Figure 5, with O-D pairs $(1, 2)$ and $(3, 4)$.
  - $d^{(1,2)} = d^{(5,6)} = 1$;
  - $\mathcal{A}_1 = \{4\}$,
  - $\mathcal{A}_2 = \{1, 2, 3, 5, 6, 7\}$,
  - upper-level variable: T_4,
  - lower-level variables: x_1, x_7, $x_2 = x_4^1 = x_3$ and $x_5 = x_4^2 = x_6$,
  - starting point: $(T_4, x_1, x_2, x_5, x_7) = (0, 0, 0, 0, 0)$.

  Again, constraints (17b) are omitted for this problem.

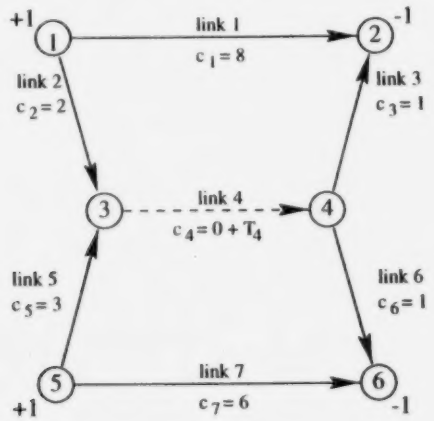

Figure 5: Network for problem `Toll5`.

Upper/lower variables in (1)	Names in (17)	BIPA	DONLP2
x_{11}	T_4	`x[0]`	`xnew[0]`
x_{21}	x_1	`x[1]`	`x[1]`
x_{22}	$x_2 = x_4^1 = x_3$	`x[2]`	`x[2]`
x_{23}	$x_5 = x_4^2 = x_6$	`x[3]`	`x[3]`
x_{24}	x_7	`x[4]`	`x[4]`

Table 11: Conversion table for the variables of problem `TOLL5`.

4.7 MPEC

The following problem (named `Outrata94P1`) was introduced by Outrata [14].

$$
\begin{array}{lll}
\min_{x_1} & F(x_1, x_2) & \\
\text{s.t.} & 0 \leq x_1 \leq 10 & \\
 & \min_{x_2} & f(x_1, x_2) \\
 & \text{s.t.} & -0.333x_{21} + x_{22} - 1 + 0.1x_1 \leq 0, \\
 & & x_{21}^2 + x_{22}^2 - 9 - 0.1x_1 \leq 0, \\
 & & -x_{21} \leq 0, \\
 & & -x_{22} \leq 0,
\end{array}
$$

where

$$f(x_1, x_2) = \frac{1}{2}\left((1 + 0.2x_1)x_{21}^2 + (1 + 0.1x_1)x_{22}^2\right) - (3 + 1.333x_1)x_{21} - x_1 x_{22},$$

and

$$F(x_1, x_2) = \frac{1}{2}(x_{21} - 3)^2 + \frac{1}{2}(x_{22} - 4.0)^2$$

Two initial values for x_1 are proposed in [14]: $x_1 = 0$ and $x_1 = 10$.

5 Example

We conclude this paper with a complete example, for which we provide all input files and results. We choose to consider problem `Bard88Ex2`, whose formulation is repeated here for convenience:

$$
\begin{array}{lll}
\min_{x_1} & F(x_1,x_2) & = -(200 - x_{21} - x_{23})(x_{21} + x_{23}) - (160 - x_{22} - x_{24})(x_{22} + x_{24}) \\
\text{s.t.} & x_{11} + x_{12} & + x_{13} + x_{14} \leq 40, \\
 & 0 \leq x_{11} \leq & 10, \\
 & 0 \leq x_{12} \leq & 5, \\
 & 0 \leq x_{13} \leq & 15, \\
 & 0 \leq x_{14} \leq & 20, \\
 & \min_{x_2} & f(x_1,x_2) = (x_{21} - 4)^2 + (x_{22} - 13)^2 + (x_{23} - 35)^2 + (x_{24} - 2)^2 \\
 & \text{s.t.} & 0.4x_{21} + 0.7x_{22} - x_{11} \leq 0, \\
 & & 0.6x_{21} + 0.3x_{22} - x_{12} \leq 0, \\
 & & 0.4x_{23} + 0.7x_{24} - x_{13} \leq 0, \\
 & & 0.6x_{23} + 0.3x_{24} - x_{14} \leq 0, \\
 & & 0 \leq x_{21} \leq 20, \\
 & & 0 \leq x_{22} \leq 20, \\
 & & 0 \leq x_{23} \leq 40, \\
 & & 0 \leq x_{24} \leq 40.
\end{array}
$$

For a better understanding of the C files reproduced hereafter and containing functions related to this problem, we provide a conversion table similar to those of problems `Toll1-Toll5` with all the notations we use in the various frameworks. This is represented in Table 12 below.

Upper/lower variables	BIPA functions (`prob_eval.c`)	DONLP2 functions (`donlp2_eval.c`)
x_{11}	`x[0]`	`xnew[0]`
x_{12}	`x[1]`	`xnew[1]`
x_{13}	`x[2]`	`xnew[2]`
x_{14}	`x[3]`	`xnew[3]`
x_{21}	`x[4]`	`x[1]`
x_{22}	`x[5]`	`x[2]`
x_{23}	`x[6]`	`x[3]`
x_{24}	`x[7]`	`x[4]`

Table 12: Conversion table for the variables of problem `Bard88Ex2`.

5.1 Parameters related to the problem: `PROBLEM.DAT`

Problem `Bard88Ex2` involves $n_1 = 4$ upper-level variables, $n_2 = 4$ lower-level variables, 9 constraints at the upper level and 12 constraints at the lower level.
The starting point is $(x_{11}, x_{12}, x_{13}, x_{14}, x_{21}, x_{22}, x_{23}, x_{24}) = (5, 5, 15, 15, 0, 0, 0, 0)$.
These data and the problem name are stored in the file `PROBLEM.DAT` as follows:

```
bard88ex2
N1 4
N2 4
M1 9
M2 12
X_11  5
X_12  5
X_13 15
X_14 15
X_21  0
X_22  0
X_23  0
X_24  0
```

Algorithmic parameters are stored in the file `SETUP.DAT`, and we use the same values as those given in Section 3.1.1.

5.2 Problem functions

From the formulation of the problem we may deduce that

$$F(x_1, x_2) = -(200 - x_{21} - x_{23})(x_{21} + x_{23}) - (160 - x_{22} - x_{24})(x_{22} + x_{24}),$$

$$G(x_1) = \begin{pmatrix} & x_{11} + x_{12} + x_{13} + x_{14} - 40 \\ - & x_{11} \\ - & x_{12} \\ - & x_{13} \\ - & x_{14} \\ & x_{11} - 10 \\ & x_{12} - 5 \\ & x_{13} - 15 \\ & x_{14} - 20 \end{pmatrix},$$

$$f(x_1, x_2) = (x_{21} - 4)^2 + (x_{22} - 13)^2 + (x_{23} - 35)^2 + (x_{24} - 2)^2,$$

$$g(x_1, x_2) = \begin{pmatrix} & 0.4x_{21} + 0.7x_{22} - x_{11} \\ & 0.6x_{21} + 0.3x_{22} - x_{12} \\ & 0.4x_{23} + 0.7x_{24} - x_{13} \\ & 0.6x_{23} + 0.3x_{24} - x_{14} \\ - & x_{21} \\ - & x_{22} \\ - & x_{23} \\ - & x_{24} \\ & x_{21} - 20 \\ & x_{22} - 20 \\ & x_{23} - 40 \\ & x_{24} - 40 \end{pmatrix}.$$

We may compute the gradients, Jacobians and Hessian matrices introduced in (7), (8), (9), (10), (11) and (12) respectively (see page 2):

$$c_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad c_2 = \begin{pmatrix} -200 + 2x_{21} + 2x_{23} \\ -160 + 2x_{22} + 2x_{24} \\ -200 + 2x_{23} + 2x_{21} \\ -160 + 2x_{24} + 2x_{22} \end{pmatrix},$$

$$d_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad d_2 = \begin{pmatrix} 2x_{21} - 8 \\ 2x_{22} - 26 \\ 2x_{23} - 70 \\ 2x_{24} - 4 \end{pmatrix},$$

$$A_1 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$H_1 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad H_2 = \begin{pmatrix} 0.4 & 0.7 & 0 & 0 \\ 0.6 & 0.3 & 0 & 0 \\ 0 & 0 & 0.4 & 0 \\ 0 & 0 & 0.6 & 0.3 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$Q_{11} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \qquad Q_{12} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$Q_{21} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \qquad Q_{22} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}.$$

These functions are written in the file `prob_eval.c`, which is reproduced below.

```
/*=========================================================================
  User-specified functions for problem "bard88ex2".

  Contains functions for evaluating upper- and lower-level functions as
  well as their gradients, Jacobians or Hessians.

  Programming: Benoit Colson (May 2002).
  =========================================================================*/

#include <stdio.h>
#include <math.h>

#include "bipa.h"

double evalFu( double *x )
{
  double eval;

  eval = - ( 200 - x[4] - x[6] ) * ( x[4] + x[6] )
    - ( 160 - x[5] - x[7] ) * ( x[5] + x[7] );

  return eval;
}

void evalGu( double *Gu, double *x )
{
  Gu[0] =   x[0] + x[1] + x[2] + x[3] - 40;
  Gu[1] = - x[0];
  Gu[2] = - x[1];
  Gu[3] = - x[2];
  Gu[4] = - x[3];
  Gu[5] =   x[0] - 10;
  Gu[6] =   x[1] -  5;
  Gu[7] =   x[2] - 15;
  Gu[8] =   x[3] - 20;

  return;
}

double evalfl( double *x )
{
  double eval;

  eval = pow( ( x[4] - 4 ), 2 ) + pow( ( x[5] - 13 ), 2 )
    + pow( ( x[6] - 35 ), 2 ) + pow( ( x[7] - 2 ), 2 );

  return eval;
}

void evalgl( double *gl, double *x )
{
  gl[0]  = 0.4 * x[4] + 0.7 * x[5] - x[0];
  gl[1]  = 0.6 * x[4] + 0.3 * x[5] - x[1];
  gl[2]  = 0.4 * x[6] + 0.7 * x[7] - x[2];
  gl[3]  = 0.6 * x[6] + 0.3 * x[7] - x[3];
  gl[4]  =     - x[4];
  gl[5]  =     - x[5];
  gl[6]  =     - x[6];
  gl[7]  =     - x[7];
  gl[8]  =       x[4] - 20;
```

```
  gl[9]  =        x[5] - 20;
  gl[10] =        x[6] - 40;
  gl[11] =        x[7] - 40;

  return;
}

void evalFugrad1( double *Fugrad1, double *x )
{
  int i;
  for( i = 0; i < 4; i++ ){
    Fugrad1[i] = 0;
  }

  return;
}

void evalFugrad2( double *Fugrad2, double *x )
{
  Fugrad2[0] = - 200 + 2 * x[4] + 2 * x[6];
  Fugrad2[1] = - 160 + 2 * x[5] + 2 * x[7];
  Fugrad2[2] = - 200 + 2 * x[6] + 2 * x[4];
  Fugrad2[3] = - 160 + 2 * x[7] + 2 * x[5];

  return;
}

void evalGujac1( double **Gujac1, double *x )
{
  Gujac1[0][0] = 1;
  Gujac1[0][1] = 1;
  Gujac1[0][2] = 1;
  Gujac1[0][3] = 1;

  Gujac1[1][0] = -1;
  Gujac1[1][1] =  0;
  Gujac1[1][2] =  0;
  Gujac1[1][3] =  0;

  Gujac1[2][0] =  0;
  Gujac1[2][1] = -1;
  Gujac1[2][2] =  0;
  Gujac1[2][3] =  0;

  Gujac1[3][0] =  0;
  Gujac1[3][1] =  0;
  Gujac1[3][2] = -1;
  Gujac1[3][3] =  0;

  Gujac1[4][0] =  0;
  Gujac1[4][1] =  0;
  Gujac1[4][2] =  0;
  Gujac1[4][3] = -1;

  Gujac1[5][0] =  1;
  Gujac1[5][1] =  0;
  Gujac1[5][2] =  0;
  Gujac1[5][3] =  0;

  Gujac1[6][0] =  0;
  Gujac1[6][1] =  1;
  Gujac1[6][2] =  0;
```

```
  Gujac1[6][3] =  0;

  Gujac1[7][0] =  0;
  Gujac1[7][1] =  0;
  Gujac1[7][2] =  1;
  Gujac1[7][3] =  0;

  Gujac1[8][0] =  0;
  Gujac1[8][1] =  0;
  Gujac1[8][2] =  0;
  Gujac1[8][3] =  1;

  return;
}

void evalGujac2( double **Gujac2, double *x )
{
  int i, j;
  for( i = 0; i < 9; i++ ){
    for( j = 0; j < 4; j++ ){
      Gujac2[i][j] = 0;
    }
  }

  return;
}

void evalflgrad1( double *flgrad1, double *x )
{
  int i;
  for( i = 0; i < 4; i++ ){
    flgrad1[i] = 0;
  }

  return;
}

void evalflgrad2( double *flgrad2, double *x )
{
  flgrad2[0] = 2 * x[4] -  8;
  flgrad2[1] = 2 * x[5] - 26;
  flgrad2[2] = 2 * x[6] - 70;
  flgrad2[3] = 2 * x[7] -  4;

  return;
}

void evalgljac1( double **gljac1, double *x )
{
  int i, j;
  for( i = 0; i < 12; i++ ){
    for( j = 0; j < 4; j++ ){
      gljac1[i][j] = 0;
    }
  }

  gljac1[0][0] = -1;
  gljac1[1][1] = -1;
  gljac1[2][2] = -1;
  gljac1[3][3] = -1;

  return;
```

```
}

void evalgljac2( double **gljac2, double *x )
{
  int i, j;
  for( i = 0; i < 12; i++ ){
    for( j = 0; j < 4; j++ ){
      gljac2[i][j] = 0;
    }
  }

  gljac2[0][0]  =  0.4;
  gljac2[0][1]  =  0.7;
  gljac2[1][0]  =  0.6;
  gljac2[1][1]  =  0.3;
  gljac2[2][2]  =  0.4;
  gljac2[2][3]  =  0.7;
  gljac2[3][2]  =  0.6;
  gljac2[3][3]  =  0.3;
  gljac2[4][0]  = -1;
  gljac2[5][1]  = -1;
  gljac2[6][2]  = -1;
  gljac2[7][3]  = -1;
  gljac2[8][0]  =  1;
  gljac2[9][1]  =  1;
  gljac2[10][2] =  1;
  gljac2[11][3] =  1;

  return;
}

void evalflhess11( double **flhess11, double *x )
{
  int i, j;
  for( i = 0; i < 4; i++ ){
    for( j = 0; j < 4; j++ ){
      flhess11[i][j] = 0;
    }
  }

  return;
}

void evalflhess12( double **flhess12, double *x )
{
  int i, j;
  for( i = 0; i < 4; i++ ){
    for( j = 0; j < 4; j++ ){
      flhess12[i][j] = 0;
    }
  }

  return;
}

void evalflhess21( double **flhess21, double *x )
{
  int i, j;
  for( i = 0; i < 4; i++ ){
    for( j = 0; j < 4; j++ ){
      flhess21[i][j] = 0;
    }
```

```
  }

  return;
}

void evalflhess22( double **flhess22, double *x )
{
  int i, j;
  for( i = 0; i < 4; i++ ){
    for( j = 0; j < 4; j++ ){
      flhess22[i][j] = 0;
    }
  }

  flhess22[0][0] = 2;
  flhess22[1][1] = 2;
  flhess22[2][2] = 2;
  flhess22[3][3] = 2;

  return;
}
```

5.3 Function evaluations for DONLP2

These functions are part of the file `donlp2_eval.c`. Recall that DONLP2 is used to compute the reaction (15) of the lower-level problem for a given vector x_1^m. In the framework of problem `Bard88Ex2`, this corresponds to solving the following problem:

$$\begin{array}{ll} \min_{x_2} & f(x_1^m, x_2) = (x_{21}-4)^2 + (x_{22}-13)^2 + (x_{23}-35)^2 + (x_{24}-2)^2 \\ \text{s.t.} & 0.4x_{21} + 0.7x_{22} - x_{11}^m \leq 0, \\ & 0.6x_{21} + 0.3x_{22} - x_{12}^m \leq 0, \\ & 0.4x_{23} + 0.7x_{24} - x_{13}^m \leq 0, \\ & 0.6x_{23} + 0.3x_{24} - x_{14}^m \leq 0, \\ & 0 \leq x_{21} \leq 20, \\ & 0 \leq x_{22} \leq 20, \\ & 0 \leq x_{23} \leq 40, \\ & 0 \leq x_{24} \leq 40, \end{array}$$

where x_1^m is a fixed vector stored in `xnew[0]`, ..., `xnew[3]` (see Table 12). This results in the following file:

```
/*=========================================================================
  User-specified functions for using DONLP2 for problem "bard88ex2".
  Contains functions for evaluating lower-level objective, constraints and
  and their gradients.

  Programming: Benoit Colson (May 2002).
  =========================================================================*/

/* IMPORTANT: using xnew as variable name for getting upper-level solution
   is compulsory. */

/* ************************************************************************ */
/*                          user functions for donlp2                       */
/* ************************************************************************ */
#include "o8para.h"
```

```
/* ************************************************************************** */
/*                              donlp2 standard setup                         */
/* ************************************************************************** */
void setup0(void) {
#define  X extern
#include "o8comm.h"

#undef   X

  extern double *xnew;

  static INTEGER j;

  strcpy(name,"lowlevel");

  x[1] = xnew[4];
  x[2] = xnew[5];
  x[3] = xnew[6];
  x[4] = xnew[7];

  n    =  4;
  nh   =  0;
  ng   = 12;
  del0 = 1.00e-1;
  tau0 = 0.5e0;
  tau  = .1e0;

  for (j = 0 ; j <= 4 ; j++) {
    gunit[1][j] = -1;
    gunit[2][j] =  0;
    gunit[3][j] =  0;
  }

  /* Special setup for bound constraints (constraints #5, #6, #7, #8 ) */
  /*  x21 >= 0 */
  gunit[1][5] =  1;
  gunit[2][5] =  1;
  gunit[3][5] =  1;

  /*  x22 >= 0 */
  gunit[1][6] =  1;
  gunit[2][6] =  2;
  gunit[3][6] =  1;

  /*  x23 >= 0 */
  gunit[1][7] =  1;
  gunit[2][7] =  3;
  gunit[3][7] =  1;

  /*  x24 >= 0 */
  gunit[1][8] =  1;
  gunit[2][8] =  4;
  gunit[3][8] =  1;

  /*  x21 <= 20 */
  gunit[1][9] =  1;
  gunit[2][9] =  1;
  gunit[3][9] = -1;

  /*  x22 <= 20 */
  gunit[1][10] =  1;
  gunit[2][10] =  2;
```

```
  gunit[3][10] = -1;

  /*  x23 <= 40 */
  gunit[1][11] =  1;
  gunit[2][11] =  3;
  gunit[3][11] = -1;

  /*  x24 <= 40 */
  gunit[1][12] =  1;
  gunit[2][12] =  4;
  gunit[3][12] = -1;

  analyt = TRUE;
  epsdif = 0.e0;
  nreset = 4;
  silent = FALSE;

  return;
}

/* ********************************************************************** */
/*                                special setup                           */
/* ********************************************************************** */
void setup(void) {
#define  X extern
#include "o8comm.h"
#undef   X

  return;
}

/* ********************************************************************** */
/* the user may add additional computations using the computed solution here */
/* ********************************************************************** */
void solchk(void) {
#define  X extern
#include "o8comm.h"
#undef   X
#include "o8cons.h"

  return;
}

/* ********************************************************************** */
/*                            objective function                          */
/* ********************************************************************** */
void ef(DOUBLE x[],DOUBLE *fx) {
#define  X extern
#include "o8fuco.h"
#undef   X

  extern double *xnew;

  icf = icf+1;

  *fx = pow( ( x[1] - 4 ), 2 ) + pow( ( x[2] - 13 ), 2 )
    + pow( ( x[3] - 35 ), 2 ) + pow( ( x[4] - 2 ), 2 );

  return;
}
```

```
/* ************************************************************************** */
/*                          gradient of objective function                    */
/* ************************************************************************** */
void egradf(DOUBLE x[],DOUBLE gradf[]) {
#define  X extern
#include "o8fuco.h"
#undef   X

  extern double *xnew;

  icgf = icgf+1;

  gradf[1] = 2 * x[1] -  8;
  gradf[2] = 2 * x[2] - 26;
  gradf[3] = 2 * x[3] - 70;
  gradf[4] = 2 * x[4] -  4;

  return;
}

/* ************************************************************************** */
/*                compute the i-th equality constaint, value is hxi           */
/* ************************************************************************** */
void eh(INTEGER i,DOUBLE x[],DOUBLE *hxi) {
#define  X extern
#include "o8fuco.h"
#undef   X

  return;
}

/* ************************************************************************** */
/*              compute the gradient of the i-th equality constraint          */
/* ************************************************************************** */
void egradh(INTEGER i,DOUBLE x[],DOUBLE gradhi[]) {
#define  X extern
#include "o8fuco.h"
#undef   X

  return;
}

/* ************************************************************************** */
/*              compute the i-th inequality constaint, bounds included        */
/* ************************************************************************** */
void eg(INTEGER i,DOUBLE x[],DOUBLE *gxi) {
#define  X extern
#include "o8fuco.h"
#undef   X

  extern double *xnew;

  cres[i] = cres[i] + 1;

  switch(i) {

  case 1:

    *gxi = - 0.4 * x[1] - 0.7 * x[2] + xnew[0];
    return;

  case 2:
```

```
      *gxi = - 0.6 * x[1] - 0.3 * x[2] + xnew[1];
      return;

    case 3:

      *gxi = - 0.4 * x[3] - 0.7 * x[4] + xnew[2];
      return;

    case 4:

      *gxi = - 0.6 * x[3] - 0.3 * x[4] + xnew[3];
      return;

    case 5:

      *gxi = x[1];
      return;

    case 6:

      *gxi = x[2];
      return;

    case 7:

      *gxi = x[3];
      return;

    case 8:

      *gxi = x[4];
      return;

    case 9:

      *gxi = - x[1] + 20;
      return;

    case 10:

      *gxi = - x[2] + 20;
      return;

    case 11:

      *gxi = - x[3] + 40;
      return;

    case 12:

      *gxi = - x[4] + 40;
      return;

    }
}

/* ************************************************************************** */
/*              compute the gradient of the i-th inequality constraint        */
/*           not necessary for bounds, but constant gradients must be set     */
/*                      here e.g. using dcopy from a data-field               */
/* ************************************************************************** */
```

```
void egradg(INTEGER i,DOUBLE x[],DOUBLE gradgi[]) {
#define  X extern
#include "o8fuco.h"
#undef   X

  extern double *xnew;

  cgres[i] = cgres[i] + 1;

  switch(i) {

  case 1:

    gradgi[1] = - 0.4;
    gradgi[2] = - 0.7;
    gradgi[3] =   0.0;
    gradgi[4] =   0.0;
    return;

  case 2:

    gradgi[1] = - 0.6;
    gradgi[2] = - 0.3;
    gradgi[3] =   0.0;
    gradgi[4] =   0.0,
    return;

  case 3:

    gradgi[1] =   0.0;
    gradgi[2] =   0.0;
    gradgi[3] = - 0.4;
    gradgi[4] = - 0.7;
    return;

  case 4:

    gradgi[1] =   0.0;
    gradgi[2] =   0.0;
    gradgi[3] = - 0.6;
    gradgi[4] = - 0.3;
    return;

  case 5:

    gradgi[1] = 1;
    gradgi[2] = 0;
    gradgi[3] = 0;
    gradgi[4] = 0;
    return;

  case 6:

    gradgi[1] = 0;
    gradgi[2] = 1;
    gradgi[3] = 0;
    gradgi[4] = 0;
    return;

  case 7:

    gradgi[1] = 0;
```

```
    gradgi[2] = 0;
    gradgi[3] = 1;
    gradgi[4] = 0;
    return;

  case 8:

    gradgi[1] = 0;
    gradgi[2] = 0;
    gradgi[3] = 0;
    gradgi[4] = 1;
    return;

  case 9:

    gradgi[1] = -1;
    gradgi[2] =  0;
    gradgi[3] =  0;
    gradgi[4] =  0;
    return;

  case 10:

    gradgi[1] =  0;
    gradgi[2] = -1;
    gradgi[3] =  0;
    gradgi[4] =  0;
    return;

  case 11:

    gradgi[1] =  0;
    gradgi[2] =  0;
    gradgi[3] = -1;
    gradgi[4] =  0;
    return;

  case 12:

    gradgi[1] =  0;
    gradgi[2] =  0;
    gradgi[3] =  0;
    gradgi[4] = -1;
    return;

  }
}

/* ************************************************************************** */
/*                          user functions (if bloc == TRUE)                  */
/* ************************************************************************** */
void eval_extern(INTEGER mode) {
#define  X extern
#include "o8comm.h"
#include "o8fint.h"
#undef   X
#include "o8cons.h"

  return;
}
```

5.4 Running BIPA and obtaining results

Once files `PROBLEM.DAT`, `prob_eval.c`, and `donlp2_eval.c` are coded as shown in the previous three sections, one may compile BIPA (see the `makefile` given with BIPA). To run BIPA, just type `bipa`. While running, BIPA completes the file `RESULTS.DAT`. In the case of our example with problem `Bard88Ex2`, running BIPA on a Sun Ultra-10/300 computer (512M RAM) gives the following results:

```
Problem name   : bard88ex2
Dimension      : n1  = 4
                 n2  = 4
Initial point  : x11 =  5.00
                 x12 =  5.00
                 x13 = 15.00
                 x14 = 15.00
                 x21 =  0.00
                 x22 =  0.00
                 x23 =  0.00
                 x24 =  0.00
=> objectives  : F   = -0.00
                 f   = 1414.00
After donlp2   : x11 =  5.00
                 x12 =  5.00
                 x13 = 15.00
                 x14 = 15.00
                 x21 =  0.49
                 x22 =  6.86
                 x23 = 25.00
                 x24 =  0.00
=> objectives  : F   = -5499.37
                 f   = 153.98
```

It.	F	f	Rho	Delta	M	Donlp2	Total
	-5499.3692	153.9846		10.0000			0.00
1	-6574.4279	64.1377	0.9425	14.0000	0	0.13	0.24
2	-6590.8978	96.1156	0.2017	14.0000	0	0.15	0.46
3	-6590.8978	96.1156	-0.3381	8.4000	1	0.14	0.69
4	-6590.8978	96.1156	-0.3381	5.0400	1	0.12	0.88
5	-6590.8978	96.1156	-0.3381	3.0240	1	0.12	1.07
6	-6594.5113	54.6900	0.1117	3.0240	0	0.14	1.26
7	-6594.5113	54.6900	-0.1439	1.8144	1	0.13	1.46
8	-6596.4785	66.7273	0.0995	1.8144	0	0.13	1.66
9	-6596.4785	66.7273	-0.1242	1.0886	1	0.13	1.84
10	-6598.4687	58.8550	0.1703	1.0886	0	0.13	2.02
11	-6598.4687	58.8550	-0.2582	0.6532	1	0.14	2.22
12	-6598.4687	58.8550	-0.0923	0.3919	1	0.12	2.40
13	-6599.6279	57.9292	0.2535	0.3919	0	0.12	2.59
14	-6599.6279	57.9292	-0.5143	0.2351	1	0.13	2.77
15	-6599.7515	57.6655	0.0914	0.2351	0	0.15	2.99
16	-6599.7515	57.6655	-0.1119	0.1411	1	0.13	3.17
17	-6599.9723	57.5288	0.3329	0.1411	0	0.13	3.37
18	-6599.9723	57.5288	-0.9961	0.0847	1	0.12	3.54
19	-6599.9723	57.5288	-0.1976	0.0508	1	0.12	3.72
20	-6599.9947	57.4761	0.2816	0.0508	0	0.13	3.91
21	-6599.9947	57.4761	-0.6445	0.0305	1	0.12	4.07
22	-6599.9950	57.4939	0.0136	0.0305	0	0.14	4.29
23	-6599.9950	57.4939	-0.0140	0.0183	1	0.13	4.49
24	-6599.9998	57.4783	0.3925	0.0183	0	0.13	4.68
25	-6599.9998	57.4783	-1.8255	0.0110	1	0.13	4.87
26	-6599.9998	57.4783	-0.6960	0.0066	1	0.13	5.06
27	-6599.9998	57.4783	-0.0105	0.0039	1	0.12	5.24

```
28 -6600.0000  57.4804   0.4141   0.0039 0  0.13  5.44
29 -6600.0000  57.4804  -2.4118   0.0024 1  0.13  5.63
30 -6600.0000  57.4804  -1.1588   0.0014 1  0.14  5.83
31 -6600.0000  57.4804  -0.0779   0.0009 1  0.13  6.03
32 -6600.0000  57.4804  -0.0779   0.0009 1  0.14  6.22
-------------------------------------------------------

Solution
 x11 =  7.36
 x12 =  3.55
 x13 = 11.64
 x14 = 17.45
 x21 =  0.91
 x22 = 10.00
 x23 = 29.09
 x24 =  0.00
 F   = -6600.00
 f   = 57.48

Total CPU time :  6.22
```


Acknowledgements

This work was partly supported by the *Ministère de la Communauté Française* (Belgium).
The author is indebted to Patrice Marcotte and Gilles Savard for having given him the opportunity to build an interesting and friendly collaboration on the occasion of three stays at the *Centre de Recherche sur les Transports*.
Finally, computational support from Serge Bisaillon and Luc Rocheleau is gratefully acknowledged.

References

[1] E. Aiyoshi and K. Shimizu. A solution method for the static constrained Stackelberg problem via penalty method. *IEEE Transactions on Automatic Control*, 29:1111–1114, 1984.

[2] J. F. Bard. Convex two-level optimization. *Mathematical Programming*, 40:15–27, 1988.

[3] D. Braess. Über ein Paradox der Verkehrsplannung. *Unternehmenstorchung*, 12:258–268, 1968.

[4] L. Brotcorne. *Approches opérationnelles et stratégiques des problèmes de trafic routier*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 1998.

[5] B. Colson, P. Marcotte, and G. Savard. A trust-region method for nonlinear bilevel programming: Algorithm and computational experience. Submitted to *Computational Optimization and Applications*, July 2002.

[6] B. Colson, P. Marcotte, G. Savard, and D. L. Zhu. Theoretical study and convergence properties of a trust-region method for nonlinear bilevel programming. Working paper, July 2002.

[7] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-region methods*. SIAM, Philadelphia, 2000.

[8] S. Dempe. A necessary and a sufficient optimality condition for bilevel programming problems. *Optimization*, 25:341–354, 1992.

[9] J. E. Falk and J. Liu. On bilevel programming, Part I : general nonlinear cases. *Mathematical Programming*, 70(1):47–72, 1995.

[10] ILOG CPLEX Division. *CPLEX User's Guide*.

[11] M. Labbé, P. Marcotte, and G. Savard. A bilevel model of taxation and its applications to optimal highway pricing. *Management Science*, 44:1595–1607, 1998.

[12] P. Marcotte. Network design problem with congestion effects : A case of bilevel programming. *Mathematical Programming*, 34:142–162, 1986.

[13] A. Migdalas. Bilevel programming in traffic planning: models, methods and challenge. *Journal of Global Optimization*, 7:381–405, 1995.

[14] J. Outrata. On optimization problems with variational inequality constraints. *SIAM Journal on Optimization*, 4(2):340–357, 1994.

[15] G. Savard. *Contribution à la programmation mathématique à deux niveaux*. PhD thesis, Ecole Polytechnique de Montréal, Université de Montréal, April 1989.

[16] K. Shimizu and E. Aiyoshi. A new computational method for Stackelberg and min-max problems by use of a penalty method. *IEEE Transactions on Systems, Man, and Cybernetics*, 11:444–449, 1981.

[17] K. Shimizu, Y. Ishizuka, and J. F. Bard. *Nondifferentiable and two-level mathematical programming*. Kluwer Academic Publishers, 1997.

[18] A. H. De Silva. *Sensitivity Formulas for Nonlinear Factorable Programming and their Application to the Solution of an Implicitly Defined Optimization Model of US Crude Oil Production*. PhD thesis, George Washington University, 1978.

[19] P. Spellucci. *DONLP2 users guide*. Technical University at Darmstadt, Department of Mathematics, 64829 Darmstadt, Germany.

[20] P. Spellucci. A new technique for inconsistent problems in the SQP method. *Math. Meth. of Oper. Res.*, 47:355–400, 1998.

[21] P. Spellucci. An SQP method for general nonlinear programs using only equality constrained subproblems. *Mathematical Programming*, 82:413–448, 1998.